Automated (AI) Planning

Planning via Constraint Satisfaction

Carmel Domshlak

Logic

Essential components

- formal language for expressing statements
- model theory/semantics for making sense of them
- proof theory/axiomatics
  for deriving new statements from old

- Originally developed for studying structure of (mathematical/philosophical) arguments,
  and identifying valid arguments.
- Currently the basis for
  - programming languages like Prolog
  - representation languages in AI (e.g., planning languages)
  - verification
  - automatic theorem proving

Logical representations of state sets

- n state variables with m values induce a state space consisting of m^n states (2^n states for n Boolean state variables)
- a language for talking about *sets of states (valuations of state variables)*: propositional logic
- logical connectives $\approx$ set-theoretical operations

Syntax of propositional logic

Let P be a set of atomic propositions ($\sim$ state variables).

1. For all $p \in P$, p is a propositional formula.
2. If ϕ is a propositional formula, then so is $\neg\phi$.
3. If ϕ and ϕ' are propositional formulae, then so is $\phi \vee \phi'$.
4. If ϕ and ϕ' are propositional formulae, then so is $\phi \wedge \phi'$.
5. The symbols $\bot$ and $\top$ are propositional formulae.

The implication $\phi \rightarrow \phi'$ is an abbreviation for $\neg\phi \vee \phi'$.
The equivalence $\phi \leftrightarrow \phi'$ is an abbreviation for $(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$.

Semantics of propositional logic

A valuation of P is a function $v : P \rightarrow \{0, 1\}$. Define the notation $v \models \phi$ for valuations v and formulae ϕ by

1. $v \models p$ if and only if $v(p) = 1$, for $p \in P$.
2. $v \models \neg\phi$ if and only if $v \not\models \phi$
3. $v \models \phi \vee \phi'$ if and only if $v \models \phi$ or $v \models \phi'$
4. $v \models \phi \wedge \phi'$ if and only if $v \models \phi$ and $v \models \phi'$
5. $v \models \top$
6. $v \not\models \bot$

Propositional logic terminology

- A propositional formula ϕ is satisfiable if there is at least one valuation v so that $v \models \phi$. Otherwise it is unsatisfiable.
- A propositional formula ϕ is valid or a tautology if $v \models \phi$ for all valuations v. We write this as $\models \phi$.
- A propositional formula ϕ is a logical consequence of a propositional formula ϕ', written $\phi' \models \phi$ if $v \models \phi$ for all valuations v with $v \models \phi'$.
- Two propositional formulae ϕ and ϕ' are logically equivalent, written $\phi \equiv \phi'$, if $\phi \models \phi'$ and $\phi' \models \phi$.

Propositional logic terminology (ctd.)

- A propositional formula that is a proposition p or a negated proposition $\neg p$ for some $p \in P$ is a literal.
- A formula that is a disjunction of literals is a clause. This includes unit clauses l consisting of a single literal, and the empty clause $\bot$ consisting of zero literals.

Normal forms: NNF, CNF, DNF

Formulae vs. sets

sets		formulae
those $\frac{2^n}{2}$ states in which p is true		$p \in P$
$E \cup F$		$E \vee F$
$E \cap F$		$E \wedge F$
$E \setminus F$	(set difference)	$E \wedge \neg F$
$\overline{E}$	(complement)	$\neg E$
the empty set $\emptyset$		$\bot$
the universal set		$\top$

question about sets	question about formulae
$E \subseteq F$?	$E \models F$?
$E \subset F$?	$E \models F$ and $F \not\models E$?
$E = F$?	$E \models F$ and $F \models E$?

Propositional Logic: Inference

- Whether $\varphi \models \psi$ is true can be tested by enumerating all different interpretations involving the propositional symbols in φ and ψ
- Bad news: exponential time as there 2^n assignments $(0/1)$ to n propositional symbols
- This time cannot be improved in worst case (unless P=NP), but approaches that run much faster in practice exist
- General idea is to combine case analysis and inference
- Exhaustive procedure above based exclusively on case analysis, even worse, deals with *full* assignments
- More about this in a few slides ...

Propositional Logic: Inference

- Whether $\varphi \models \psi$ is true can be tested by enumerating all different interpretations involving the propositional symbols in φ and ψ
- Bad news: exponential time as there 2^n assignments $(0/1)$ to n propositional symbols
- This time cannot be improved in worst case (unless P=NP), but approaches that run much faster in practice exist
- General idea is to combine case analysis and inference
- Exhaustive procedure above based exclusively on case analysis, even worse, deals with *full* assignments
- More about this in a few slides ...

Conjunctive Normal Form (CNF) and SAT

Let P be a set of propositional symbols. A propositional formula Φ is called a CNF if it has the form

$$\Phi = \varphi_1 \wedge \cdots \wedge \varphi_m$$

where each φ_i has the form $\phi_i = (l_1 \vee \cdots \vee l_k)$ and each l_j is a literal over P

- in other words, a conjunction of disjunctions of literals
- why called "normal form"?

CNF $\rightsquigarrow$ formula $==$ a set of constraints

- in CNFs, each constraint φ_i is called a clause, each clause being a set of literals

SAT is the decision problem of determining whether a given CNF formula is satisfiable

Conjunctive Normal Form (CNF) and SAT

Let P be a set of propositional symbols. A propositional formula Φ is called a CNF if it has the form

$$\Phi = \varphi_1 \wedge \cdots \wedge \varphi_m$$

where each φ_i has the form $\phi_i = (l_1 \vee \cdots \vee l_k)$ and each l_j is a literal over P

- in other words, a conjunction of disjunctions of literals
- why called "normal form"?

CNF $\rightsquigarrow$ formula $==$ a set of constraints

- in CNFs, each constraint φ_i is called a clause, each clause being a set of literals

SAT is the decision problem of determining whether a given CNF formula is satisfiable

Constraint Propagation

- Given a set Φ of constraints over variables (e.g., clauses over propositional variables), infer new constraints
- Inference: some reasoning (= proof theory) R that is sound
  - if R infers φ from Φ, then $\Phi \models \varphi$
- $\Phi \cup \{\varphi\}$ is logically equivalent to Φ ... but $\Phi \cup \{\varphi\}$ can be "more informative"
  - e.g., there may be constraints ψ that R can infer in one step from $\Phi \cup \{\varphi\}$, but not from Φ
- Typically one computes a fixpoint: propagation

Resolution

Given clauses $\varphi' = \varphi \cup \{p\}$ and $\psi' = \psi \cup \{\neg p\}$, we allow the inference

$$\frac{\varphi \cup \{p\} \qquad \psi \cup \{\neg p\}}{\varphi \vee \psi}$$

That is, $\varphi \vee \psi$ can be added as a **new clause**

- Since p and $\neg p$ cannot be simultaneously true, we have to make true at least one of φ and ψ
- Resolution is complete: Φ is unsatisfiable iff $\{\} \in R^+(\Phi)$

k-Resolution and Unit Propagation

- A full (complete) constraint propagation is exponentially costly: it solves the original decision problem
- We need more restricted reasoning that will still give us some information/simplification
- k-resolution: in

$$\frac{\varphi \cup \{p\} \qquad \psi \cup \{\neg p\}}{\varphi \vee \psi}$$

  require that either $|\varphi \cup \{p\}| \leq k$ or $|\psi \cup \{\neg p\}| \leq k$
- Unit propagation $==$ 1-resolution is the most wide-spread techniques in implemented SAT solvers

Unit Propagation

Fixpoint application of

$$\frac{\varphi \cup \{\bar{l}\} \qquad \{l\}}{\varphi}$$

Procedure unit-propagation

while TRUE **do**

  $\Phi' := \Phi$

  forall $\psi \in \Phi$, $\psi = \{l\}$ **do**

    forall $\phi \in \Phi$, $\bar{l} \in \phi$ **do**

      $\Phi' := \Phi' \cup \{\phi \setminus \{\bar{l}\}\}$

  if $\Phi' = \Phi$ **then** stop

  $\Phi := \Phi'$

Unit Propagation

Procedure unit-propagation

while TRUE **do**
    $\Phi' := \Phi$
    forall $\psi \in \Phi$, $\psi = \{l\}$ **do**
        forall $\phi \in \Phi$, $\bar{l} \in \phi$ **do**
            $\Phi' := \Phi' \cup \{\phi \setminus \{\bar{l}\}\}$
            $\Phi' := \Phi' \setminus \phi$
        forall $\varphi \in \Phi'$, $l \in \varphi$ **do**
            $\Phi' := \Phi' \setminus \varphi$
    if $\Phi' = \Phi$ **then** stop
    $\Phi := \Phi'$

Unit Propagation

Procedure unit-propagation

while TRUE **do**

$\Phi' := \Phi$

forall $\psi \in \Phi$, $\psi = \{l\}$ **do**

forall $\phi \in \Phi$, $\bar{l} \in \phi$ **do**

$\Phi' := \Phi' \cup \{\phi \setminus \{\bar{l}\}\}$

$\Phi' := \Phi' \setminus \phi$

forall $\varphi \in \Phi'$, $l \in \varphi$ **do**

$\Phi' := \Phi' \setminus \varphi$

if $\Phi' = \Phi$ **then** stop

$\Phi := \Phi'$

Examples

- $\{\{\neg A, \neg B, \neg C, D\}, \{\neg A, B\}, \{A\}, \{\neg A, \neg B, \neg C, \neg D\}, \{\{\neg A, \neg B, C\}\}\}$
- $\{\{\neg A, B\}, \{\neg B, C\}, \{\neg C, A\}, \{A, C\}, \{\neg B, \neg C\}\}$

Backtracking search

Backtracking over variable values

Procedure backtracking-search

bool **Solve** (Φ, partial assignment ω)
    $(\Phi', \omega') :=$ *constraint-propagation*(Φ, ω)
    if Φ' is self-contradictory **then return** FALSE
    select a variable v not assigned by ω'
    if no such variable exists **then return** TRUE
    forall $c \in dom(v)$ **do**
        if Solve$(\Phi', \omega' \cup \{v := c\})$ **then return** TRUE
    return FALSE

Davis-Putnam-Logeman-Loveland Algorithm (DPLL)

Procedure DPLL

bool **DPLL** (Φ, partial assignment ω)

$(\Phi', \omega') :=$ *unit-propagation*(Φ, ω)

if Φ' contains empty clause **then return** FALSE

select a variable v not assigned by ω'

if no such variable exists **then return** TRUE

if DPLL$(\Phi', \omega' \cup \{v := 1\})$ **then return** TRUE

if DPLL$(\Phi', \omega' \cup \{v := 0\})$ **then return** TRUE

return FALSE

Davis-Putnam-Logeman-Loveland Algorithm (DPLL)

Procedure DPLL

bool **DPLL** (Φ, partial assignment ω)
  $(\Phi', \omega') :=$ *unit-propagation*(Φ, ω)
  if Φ' contains empty clause **then return** FALSE
  select a variable v not assigned by ω'
  if no such variable exists **then return** TRUE
  if DPLL$(\Phi', \omega' \cup \{v := 1\})$ **then return** TRUE
  if DPLL$(\Phi', \omega' \cup \{v := 0\})$ **then return** TRUE
  return FALSE

Examples

- $\{\{A, B, C\}, \{\neg A, \neg B\}, \{\neg A, \neg C\}, \{\{\neg B, \neg C\}\}\}$
- $\{\{\neg A, B\}, \{\neg B, C\}, \{\neg C, A\}, \{A, C\}, \{\neg B, \neg C\}\}$

DPLL these days (DPLL++)

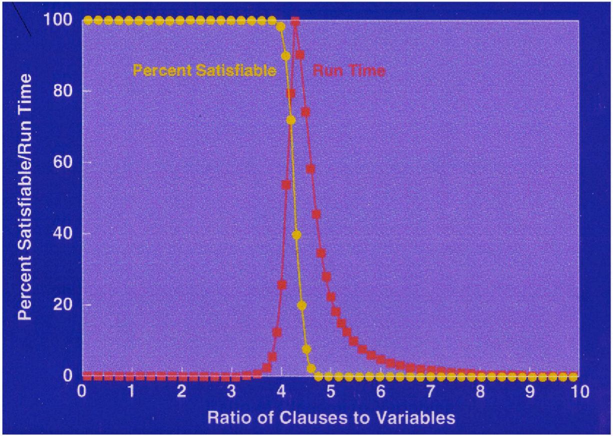
- currently very large SAT problems can be solved
- criterion for **variable selection** is critical
- additional key components
  - randomization (in selection) + restarts (???)
  - clause learning (...)
  - engineering issues (e.g., caching)
- from 50 variables, 200 constraints in early 90's to 1000000 variables and 5000000 constraints these days (from 10^{15} to 10^{300000})

Progress of SAT solvers

Instance	Posit' 94	Grasp' 96	Sato' 98	Chaff' 01
ssa2670-136	40,66s	1,2s	0,95s	0,02s
bf1355-638	1805,21s	0,11s	0,04s	0,01s
pret150_25	>3000s	0,21s	0,09s	0,01s
dubois100	>3000s	11,85s	0,08s	0,01s
aim200-2_0-no-1	>3000s	0,01s	0s	0s
2dlx_..._bug005	>3000s	>3000s	>3000s	2,9s
c6288	>3000s	>3000s	>3000s	>3000s

(Marques Silva, 02)

Phase Transition and Computational Hardness

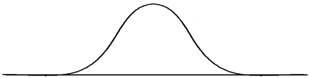


(Selman, Levesque, and David Mitchell, 92)

Pathology of backtracking search

Backtrack-style search on hard problems characterized by:

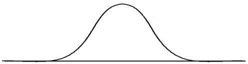
- Erratic behavior of time complexity distribution
- Distributions have **“heavy tails”**
    - infinite mean ? infinite variance ?



Standard Distribution
(finite mean & variance)

HEAVY TAILED DISTRIBUTION
(infinite mean & variance)

Idea: Randomized Restarts



Standard Distribution
(finite mean & variance)

HEAVY TAILED DISTRIBUTION
(infinite mean & variance)

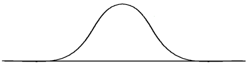
Randomize the backtrack strategy

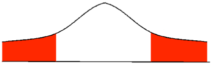
- **add noise** to the heuristic branching (variable choice) function
- **cutoff** and **restart** search after a fixed number of backtracks
  - critical parameter: cutoff threshold

Works?

- provably eliminates heavy tails
- practice: rapid restarts with low cutoff can dramatically improve performance (Gomes and Selman 1998, 1999)
- exploited in most (all?) current SAT solvers

Idea: Randomized Restarts


Standard Distribution
(finite mean & variance)


HEAVY TAILED DISTRIBUTION
(infinite mean & variance)

Randomize the backtrack strategy

- **add noise** to the heuristic branching (variable choice) function
- **cutoff** and **restart** search after a fixed number of backtracks
  - critical parameter: cutoff threshold

Works?

- provably eliminates heavy tails
- practice: rapid restarts with low cutoff can dramatically improve performance (Gomes and Selman 1998, 1999)
- exploited in most (all?) current SAT solvers

Planning via SAT: Motivation and idea

Motivation observation

- solvers are developed for many NP-complete classes of problems
- progress is not uniform (reasons?)
- progress in solving SAT is probably most prominent

Idea (Kautz & Selman, 91-96)

- Maybe we should teach SAT solvers to solve planning?
- Problem: Strips planning is PSPACE-complete
- Solution: Bounded-Strips planning is in NP

Planning via SAT: Motivation and idea

Motivation observation

- solvers are developed for many NP-complete classes of problems
- progress is not uniform (reasons?)
- progress in solving SAT is probably most prominent

Idea (Kautz & Selman, 91-96)

- Maybe we should teach SAT solvers to solve planning?
- Problem: Strips planning is PSPACE-complete
- Solution: Bounded-Strips planning is in NP

Planning as Satisfiability

Transform Planning into a series of SATs

Procedure planning-as-SAT($\Pi = (P, A, I, G)$)

$b = 0$

while TRUE **do**

  $\Phi(\Pi, b)$:= a CNF that is satisfiable iff there exists a plan with b steps

  if DPLL($\Phi(\Pi, b), \emptyset$) **then**

    output Plan encoded by a satisfying assignment

    $b := b + 1$

Questions

- What notions of "steps" can we use?
- What do we know about the found plan?
- What should be the connection between the set of plans for Π and the set of satisfying assignments to $\Phi(\Pi, b)$?
- What can we say about the completeness of the algorithm?

Strips encodings

How to encode b-step Strips plan existence as a CNF?

Many possible answers. Most (in use to date) share:

- *Time steps* $0 \leq t \leq b$
- Fact variables p_t: is p TRUE or FALSE at t?
- Action variables a_t: is a applied at t or not?

- The size of the encoding grows linearly in b
  - but is it a linear grows in the size of the input?

The Linear Encoding, I
Sequential planning

- Problem $\Pi = (P, A, I, G)$, time steps $0 \leq t \leq b$
- Decision variables

  p_t — for all $p \in P, 0 \leq t \leq b$

  a_t — for all $a \in A, 0 \leq t \leq b-1$
- Initial State Clauses: "specify initial state"

  for all $p \in P$: $\{p_0\}$ if $p \in I$, and $\{\neg p_0\}$, otherwise
- Goal Clauses: "specify goal values"

  for all $p \in G$: $\{p_b\}$

The Linear Encoding, II
Sequential planning

- Action Precondition Clauses:
  "action implies its preconditions"

  for all $a \in A, p \in pre(a), 0 \leq t \leq b-1$: $\{\neg a_t, p_t\}$

- Action Effect Clauses:
  "action implies its add/delete effects"

  for all $a \in A, p \in add(a), 0 \leq t \leq b-1$: $\{\neg a_t, p_{t+1}\}$

  for all $a \in A, p \in del(a), 0 \leq t \leq b-1$: $\{\neg a_t, \neg p_{t+1}\}$

The Linear Encoding, III

Sequential planning

- Positive Frame Axioms:
  "if a is applied and $p \notin$ del(a) was true, then p is still true"

  for all $a \in A, p \notin$ del$(a), 0 \leq t \leq b-1$: $\{\neg a, \neg p_t, p_{t+1}\}$

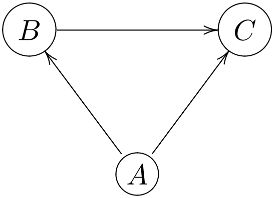
- Negative Frame Axioms:
  "if a is applied and $p \notin$ add(a) was false, then p is still false"

  for all $a \in A, p \notin$ add$(a), 0 \leq t \leq b-1$: $\{\neg a, p_t, \neg p_{t+1}\}$

- Linearity (Exclusion) Constraints:
  "apply exactly one action at each time step"

  for all $a, a' \in A, 0 \leq t \leq b-1$: $\{\neg a, \neg a'_t\}$
  for all $0 \leq t \leq b-1$: A_t (*do we really need them?*)

Example



- $P = \{A, B, C, visB, visC\}$, $I = \{A\}$, $G = \{visB, visC\}$
- Actions

$$
\begin{aligned}
drAB &= \{\{A\}, \{B, visB\}, \{A\}\} \\
drAC &= \{\{A\}, \{C, visC\}, \{A\}\} \\
drBC &= \{\{B\}, \{C, visC\}, \{B\}\}
\end{aligned}
$$

Blackboard: Linear encoding for $b = 1$

A Basic Parallel Encoding, I

Parallel planning

- Problem $\Pi = (P, A, I, G)$, noops-extended actions A^N, time steps $0 \leq t \leq b$
- Decision variables

  p_t — for all $p \in P, 0 \leq t \leq b$

  a_t — for all $a \in A^N, 0 \leq t \leq b-1$
- Initial State Clauses: "specify initial state"

  for all $p \in P$: $\{p_0\}$ if $p \in I$, and $\{\neg p_0\}$, otherwise
- Goal Clauses: "specify goal values"

  for all $p \in G$: $\{p_b\}$

A Basic Parallel Encoding, II

Parallel planning

- Action Precondition Clauses:
  "action implies its preconditions"

  for all $a \in A^N, p \in pre(a), 0 \leq t \leq b-1$: $\{\neg a_t, p_t\}$

- Action Interference Clauses:
  "do not apply interfering actions in the same time step"

  for all $a, a' \in A^N, a \not\| a', 0 \leq t \leq b-1$: $\{\neg a_t, \neg a'_t\}$

- Fact Achievement Clauses:
  "fact implies disjunction of its achievers"

  for all $p \in P, 1 \leq t \leq b$: $\{\neg p_t\} \cup \{a_{t-1} | p \in \text{add}(a)\}$

Do we need anything else?

A Basic Parallel Encoding, II
Parallel planning

- Action Precondition Clauses:
  "action implies its preconditions"
  for all $a \in A^N, p \in pre(a), 0 \leq t \leq b-1$: $\{\neg a_t, p_t\}$
- Action Interference Clauses:
  "do not apply interfering actions in the same time step"
  for all $a, a' \in A^N, a \not| a', 0 \leq t \leq b-1$: $\{\neg a_t, \neg a'_t\}$
- Fact Achievement Clauses:
  "fact implies disjunction of its achievers"
  for all $p \in P, 1 \leq t \leq b$: $\{\neg p_t\} \cup \{a_{t-1} | p \in \text{add}(a)\}$

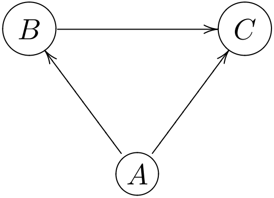
Do we need anything else?

Linear vs. Parallel Encodings

- Optimal parallel plans are often shorter than optimal sequential plans
- Linearity constraints typically dominate the linear encodings

So in parallel planning-as-SAT we (typically) need fewer iterations and (always) consider smaller formulas!

Example



- $P = \{A, B, C, visB, visC\}$, $I = \{A\}$, $G = \{visB, visC\}$
- Actions

$$drAB = \{\{A\}, \{B, visB\}, \{A\}\}$$
$$drAC = \{\{A\}, \{C, visC\}, \{A\}\}$$
$$drBC = \{\{B\}, \{C, visC\}, \{B\}\}$$

Blackboard: Basic parallel encoding for $b = 1$

2-Planning Graphs

2-planning graphs extend 1-planning graphs by keeping track of **mutex pairs**; pairs that cannot be **simultaneously** achieved in i steps:

- **action pair mutex** at i if actions interfere or their preconditions mutex at i
- **atom pair mutex** at i if all supporting action pairs are mutex at $i-1$
- a **set of atoms** C is **mutex** at i if it contains a mutex pair at i

Resulting graph:

- $P_0 = \{p \in I\}$
- $A_i = \{a \in A^N \mid Prec(a) \subseteq P_i \text{ and not mutex at } i\}$
- $P_{i+1} = \{p \in Add(a) \mid a \in A_i\}$,
  with sets of action/atom mutex pairs defined as above.

The Planning Graph Based Encoding, I

- Problem $\Pi = (P, A, I, G)$, noops-extended actions A^N, time steps $0 \leq t \leq b$
- Fact layers $P_{(t)}$, action layers $A_{(t)}$, fact mutexes (layers) $EP_{(t)}$, action mutexes (layers) $EA_{(t)}$
- Decision variables

  p_t — for all $p \in P, 1 \leq t \leq b$
  a_t — for all $a \in A^N, 0 \leq t \leq b-1$
- Goal Clauses: "specify goal values"

  for all $p \in G$: $\{p_b\}$
- Action Precondition Clauses:
  "action implies its preconditions"

  for all $a \in A^N, p \in pre(a), 1 \leq t \leq b-1$: $\{\neg a_t, p_t\}$

The Planning Graph Based Encoding, II

- Action Mutex Clauses: "do not apply mutex actions in the same time step"

  for all $0 \leq t \leq b-1, a, a' \in A_{(t)}, \{a, a'\} \in EA_{(t)}$: $\{\neg a_t, \neg a'_t\}$

- Fact Achievement Clauses:
  "fact implies disjunction of its achievers"

  for all $p \in P, 1 \leq t \leq b$: $\{\neg p_t\} \cup \{a_{t-1} | p \in \text{add}(a)\}$

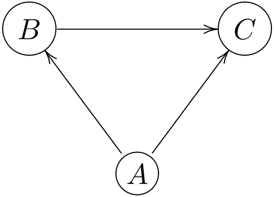
- Fact Mutex Clauses:
  "do not make two mutex facts TRUE"

  for all $1 \leq t \leq b, p, p' \in P_{(t)}, \{p, p'\} \in EP_{(t)}$: $\{\neg p_t, \neg p'_t\}$

Basic Parallel vs. PG-Based Encoding, I

- PG-Based Encoding $==$ Basic Parallel Encoding pruned and enhanced by information contained in 2-Planning Graph
- Pruned: **less** decision variables p_t and a_t, less redundant exclusion clauses
  - Example: We don?t need vars for the initial facts since pre$(a) \subseteq I$ holds anyway for all $a \in A_{(0)}$
- Enhanced: **more** non-trivial (temporal) exclusion clauses $\{\neg a_t, \neg a'_t\}$ and $\{\neg p_t, \neg p'_t\}$

Example



- $P = \{A, B, C, visB, visC\}$, $I = \{A\}$, $G = \{visB, visC\}$
- Actions

$$drAB = \{\{A\}, \{B, visB\}, \{A\}\}$$
$$drAC = \{\{A\}, \{C, visC\}, \{A\}\}$$
$$drBC = \{\{B\}, \{C, visC\}, \{B\}\}$$

Blackboard: PG-based encoding for $b = 1$

Basic Parallel vs. PG-Based Encoding, I (Recall)

- PG-Based Encoding $==$ Basic Parallel Encoding pruned and enhanced by information contained in 2-Planning Graph
- Pruned: **less** decision variables p_t and a_t, less redundant exclusion clauses
  - Example: We don?t need vars for the initial facts since pre$(a) \subseteq I$ holds anyway for all $a \in A_{(0)}$
- Enhanced: **more** non-trivial (temporal) exclusion clauses $\{\neg a_t, \neg a'_t\}$ and $\{\neg p_t, \neg p'_t\}$

Basic Parallel vs. PG-Based Encoding, II

- All new clauses (the pruned $\{\neg p_t\}$ and $\{\neg a_t\}$, and all new exclusion clauses) **follow** from the Basic Parallel CNF Φ
- By constructing 2-planning graph and basic our SAT encoding on it ...
  - ... we do some of the reasoning devoted to the SAT solver with a specialized algorithm instead
  - But why this part of work and not all the work?
- Potentially exponential savings
  - suppose (since) the SAT solver uses, in constraint propagation, 1-Resolution only
  - for exclusion relations we need 2-Resolution! [Brafman, JAIR-2001]
- *What sort of resolution do we need to capture k-planning graphs in the constraint propagation procedure?*

Basic Parallel vs. PG-Based Encoding, II

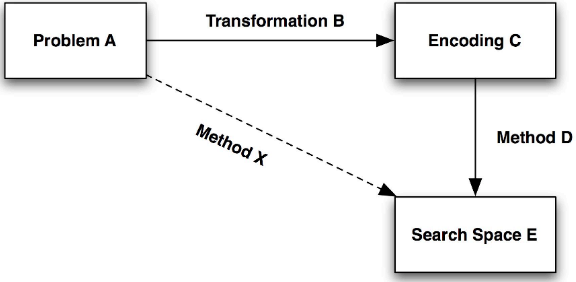
- All new clauses (the pruned $\{\neg p_t\}$ and $\{\neg a_t\}$, and all new exclusion clauses) **follow** from the Basic Parallel CNF Φ

- By constructing 2-planning graph and basic our SAT encoding on it ...
  - ... we do some of the reasoning devoted to the SAT solver with a specialized algorithm instead
  - But why this part of work and not all the work?

- Potentially exponential savings
  - suppose (since) the SAT solver uses, in constraint propagation, 1-Resolution only
  - for exclusion relations we need 2-Resolution! [Brafman, JAIR-2001]

- *What sort of resolution do we need to capture k-planning graphs in the constraint propagation procedure?*

In Front of the Curtains



- What are A, B, C, D, E in our case?
- What is X?

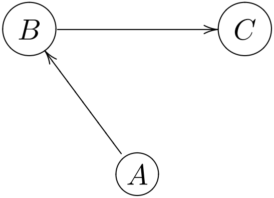
A Very Simple Encoding

Use a 1-planning graph

- Problem $\Pi = (P, A, I, G)$, noops-extended actions A^N, time steps $0 \leq t \leq b$, action layers $A_{(t)}$
- Decision variables: a_t — for all $0 \leq t \leq b-1$ and $a \in A_{(t)}$
- Goal Clauses: "at least one achiever"
  - for all $p \in G$: $\{a_{b-1} | a \in A_{(b-1)}, g \in \text{add}(a)\}$
- Action Precondition Clauses:
  "action implies disjunction of its precondition achievers"

  for all $1 \leq t \leq b-1, a \in A_{(t)}, p \in pre(a)$:
  $\{\neg a_t\} \cup \{a'_{t-1} | a' \in A_{(t-1)}, p \in \text{add}(a')\}$
- Action Interference Clauses: as in basic parallel encoding

Example



- $P = \{A, B, C\}$, $I = \{A\}$, $G = \{C\}$
- Actions

$$\begin{aligned} drAB &= \{\{A\}, \{B\}, \{A\}\} \\ drBC &= \{\{B\}, \{C\}, \{B\}\} \end{aligned}$$

Blackboard: “Very simple” encoding for $b = 2$

Reminder: DPLL

Procedure DPLL

bool **DPLL** (Φ, partial assignment ω)

$(\Phi', \omega') :=$ *unit-propagation*(Φ, ω)

if Φ' contains empty clause **then return** FALSE

select a variable v not assigned by ω'

if no such variable exists **then return** TRUE

if DPLL$(\Phi', \omega' \cup \{v := 1\})$ **then return** TRUE

if DPLL$(\Phi', \omega' \cup \{v := 0\})$ **then return** TRUE

return FALSE

Behind the Curtains, Unit Propagation, I

propagate $a_t = \text{TRUE}$

set a IN at t
if $t > 0$ **then forall** $p \in \text{pre}(a)$
    if all $a' \in A_{(t-1)}, p \in \text{add}(a')$ are OUT at $t-1$ **then fail**
    if all $a' \in A_{(t-1)}, p \in \text{add}(a')$ are OUT at $t-1$, except a''
        then propagate a'' IN at $t-1$
forall $a' \in A_{(t)}$ that interfere with a
    propagate a' OUT at t

Behind the Curtains, Unit Propagation, II

propagate a_t = FALSE

set a OUT at t
if $t = b - 1$ **then forall** $g \in$ add$(a) \cap G$
    if all $a' \in A_{(t)}, g \in$ add(a') are OUT at t **then fail**
    if all $a' \in A_{(t)}, g \in$ add(a') are OUT at t, except a''
        then propagate a'' IN at $t - 1$
if $t < b - 1$ **then**
        ???

Behind the Curtains, DPLL

- DPLL makes commitments of the form
  "I will/won't apply action a at time t"
- The search state is a sequence of such commitments

d0 "I will move the truck from x to y at time 17"

d1 UP: "truck at x at time 17", "truck at y at time 18"

d1 "I will sell the truck at time 7"

d2 UP: "no truck at time $8, \ldots, 25$"

d2 FALSE

d1 "I will not sell the truck at time 7"

- The order of commitments in the sequence is independent of the time steps t
- ... this is why we also call this undirected search

Behind the Curtains, DPLL

- DPLL makes commitments of the form "I will/won't apply action a at time t"
- The search state is a sequence of such commitments

d0 "I will move the truck from x to y at time 17"

d1 UP: "truck at x at time 17", "truck at y at time 18"

d1 "I will sell the truck at time 7"

d2 UP: "no truck at time $8, \ldots, 25$"

d2 FALSE

d1 "I will not sell the truck at time 7"

- The order of commitments in the sequence is independent of the time steps t
- ... this is why we also call this undirected search

Branching in Planning: A Big Picture

- **Forward**: state-space; extend plan head, totally (possibly weakly) ordered
- **Backward**: regression-space; extend plan tail; totally (possibly weakly) ordered
- **Temporal**: for action a and time i, create splits $a[i] = \text{TRUE}$ / $a[i] = \text{FALSE}$
- **POCL**: Partial Order Causal Link Planning
  - next ...